LES10B
Een aantal GUI-extra's

Werken met meerdere Frames
Werken met meerdere Panels

`JOptionPane`-GUI klasse
`JList`-GUI klasse

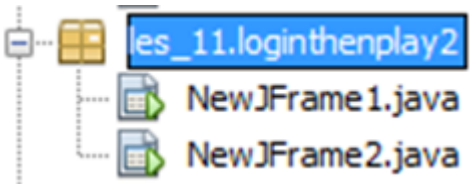
Dynamisch werken met grafische componenten

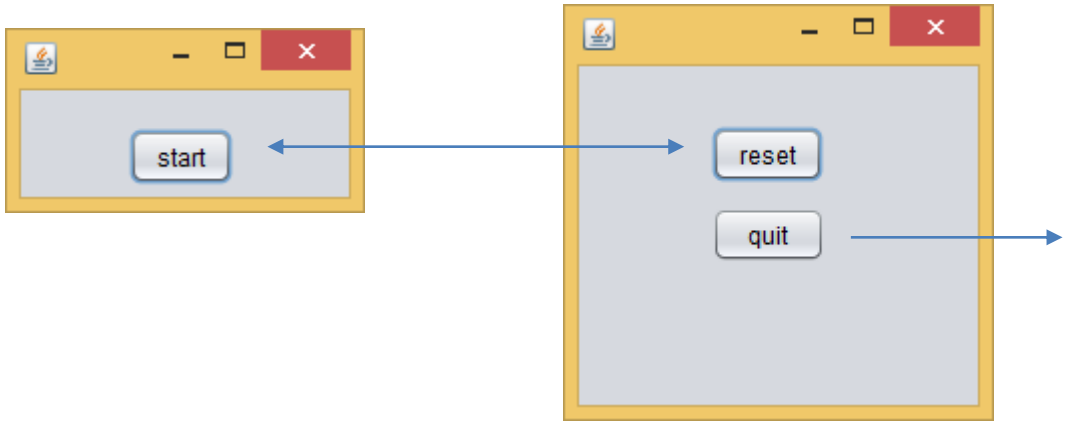
Event handling via code

Layouts

Werken met meerdere frames

2 JFrames, sequentieel

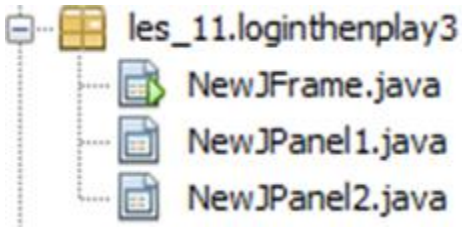


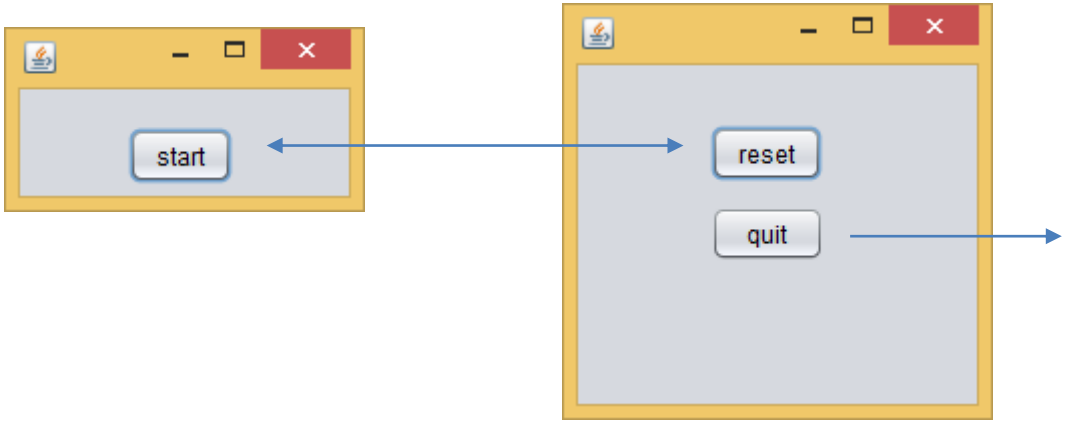



```
//in NewJFrame1
    private void jButtonStartActionPerformed(java.awt.event.ActionEvent evt)
        new NewJFrame2().setVisible(true);
        this.dispose();
    }
```

```
//in NewJFrame2
private void jButtonResetActionPerformed(java.awt.event.ActionEvent evt)
        new NewJFrame1().setVisible(true);
        this.dispose();
    }
```

1 JFrame, 2 JPanels






```
public static void main(String args[]) {
        ...
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                NewJFrame frame = new NewJFrame();
                frame.setContentPane(new NewJPanel1(frame));
                frame.setVisible(true);
            }
        });
    }
```

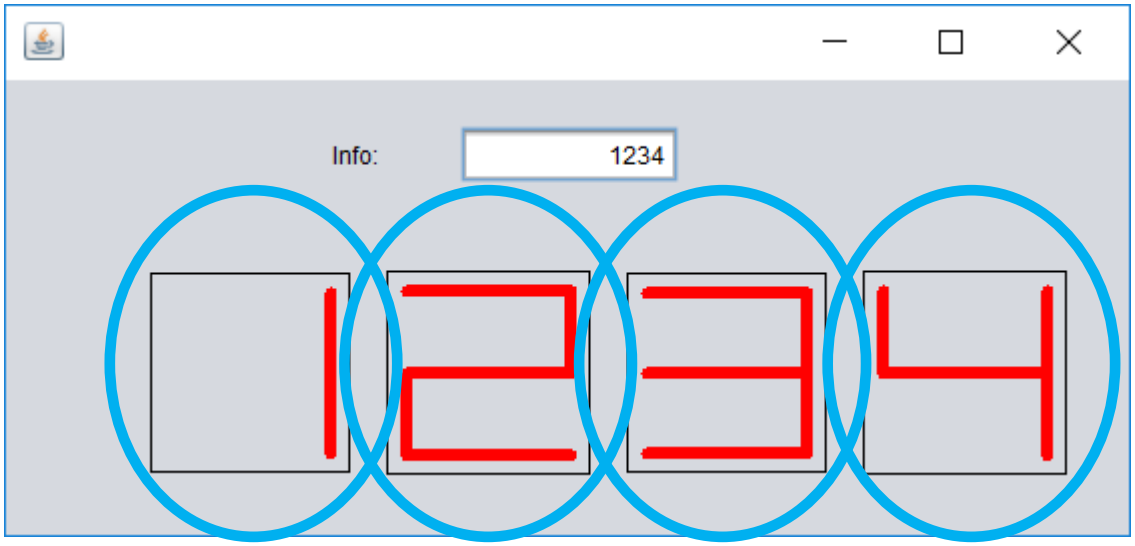
NewJPanel1

```
public class NewJPanel1 extends javax.swing.JPanel {
    private JFrame parentFrame;

    public NewJPanel1(JFrame frame) {
        initComponents();
        this.parentFrame = frame;
    }

    …

    private void jButtonStartActionPerformed(java.awt.event.ActionEvent evt)
        parentFrame.setContentPane(new NewJPanel2(parentFrame));
        parentFrame.validate();
    }


    // Variables declaration - do not modify
    private javax.swing.JButton jButtonStart;
    // End of variables declaration
}
```

NewJPanel2

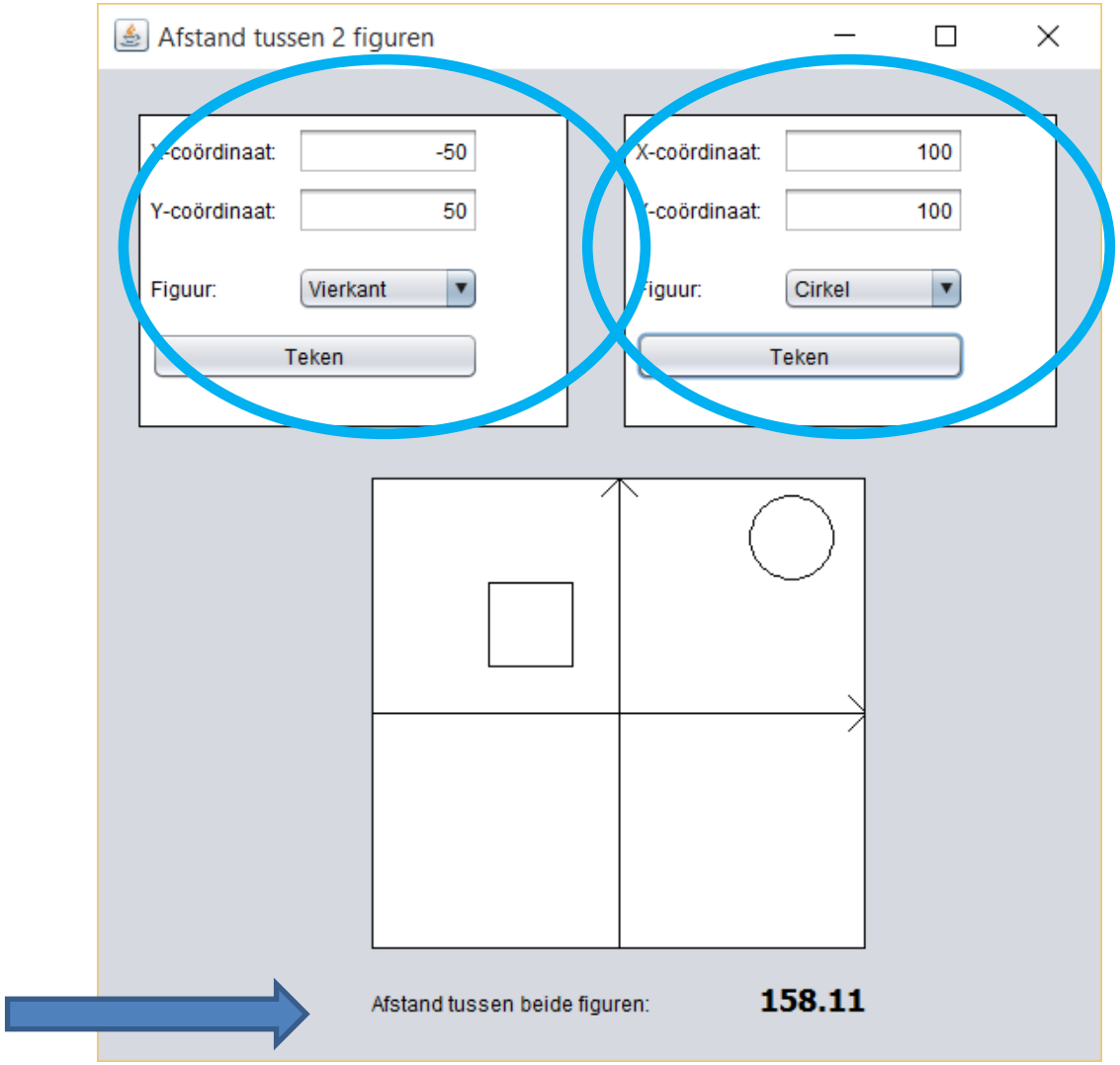
```
public class NewJPanel2 extends javax.swing.JPanel {
    private JFrame parentFrame;

    public NewJPanel2(JFrame frame) {
        initComponents();
        this.parentFrame = frame;
    }
    …

    private void jButtonResetActionPerformed(java.awt.event.ActionEvent evt)
        parentFrame.setContentPane(new NewJPanel1(parentFrame));
        parentFrame.validate();
    }

    private void jButtonQuitActionPerformed(java.awt.event.ActionEvent evt) {
        parentFrame.dispose();
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButtonQuit;
    private javax.swing.JButton jButtonReset;
    // End of variables declaration
}
```

Werken met zichtbare set van panels

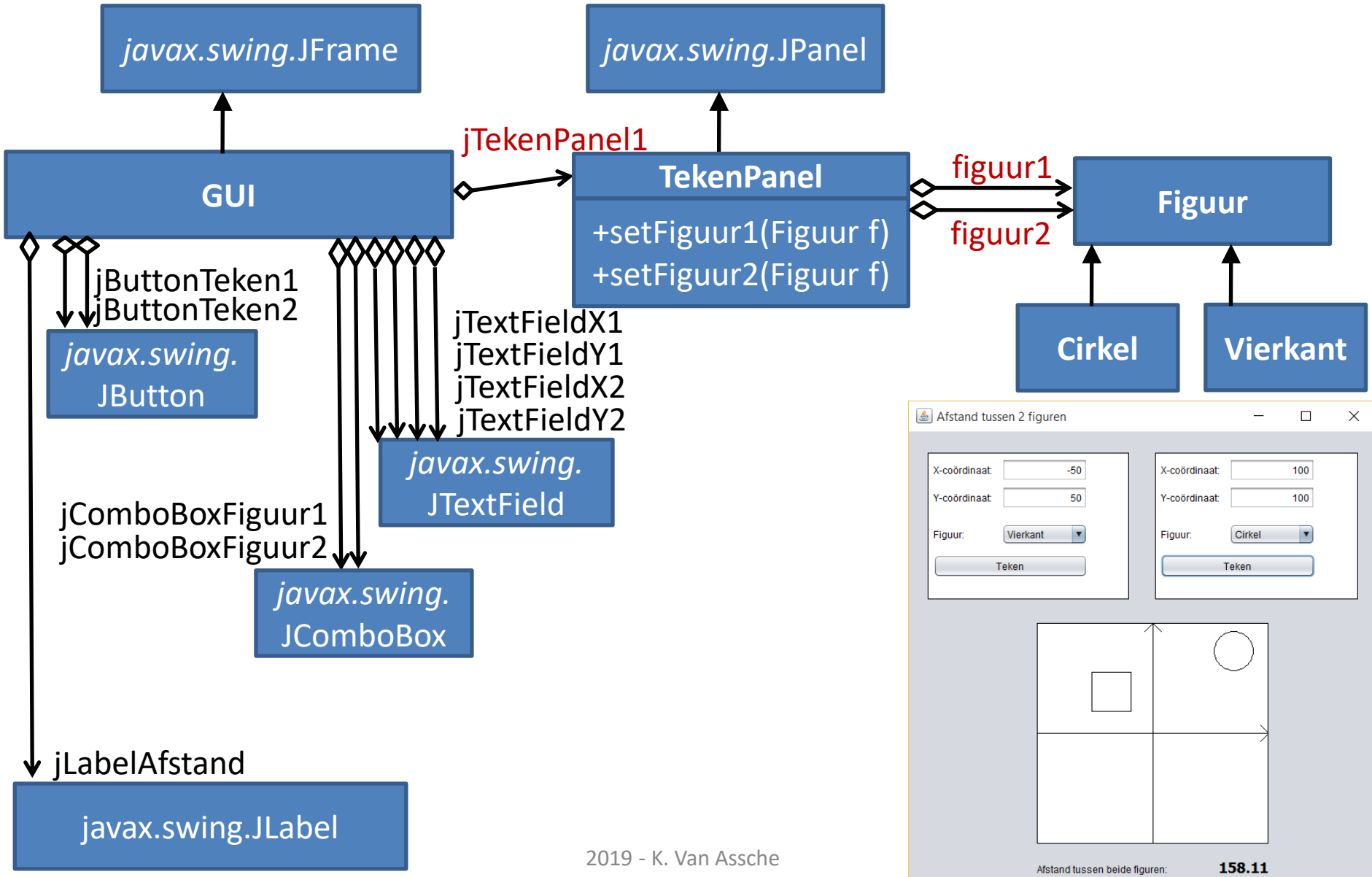


Simulatie van een 7-segment display

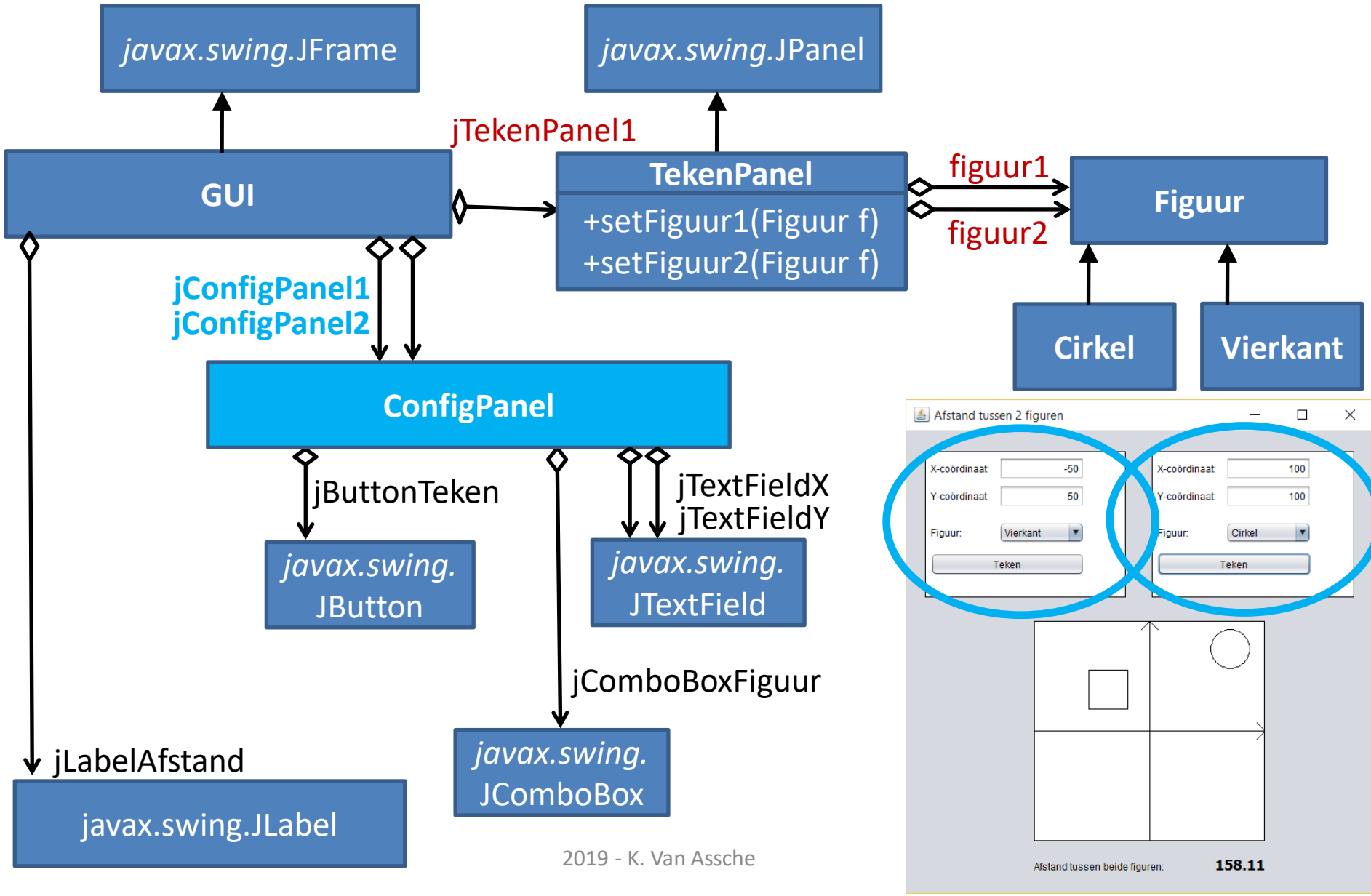
Labo 09b (vereenvoudigde GUI)



Cf. Labo09b – optie1



Cf. Labo09b – optie2



Laat configPanel object zijn nieuwe figuur doorgeven aan tekenPanel object

In ConfigPanel klasse:

```
private void jButtonVerwerkActionPerformed(java.awt.event.ActionEvent evt) {
        …
        if (figuur != null) {
            if (positie == 'l') this.tekenPanel.setFiguur1(this.figuur);
            else if (positie == 'r') this.tekenPanel.setFiguur2(this.figuur);
        }
}
```

Noot:
GUI heeft er bij initialisatie voor gezorgd dat tekenPanel ref gekend is in beide configPanels:

```
public GUI() {
    initComponents();
    this.configPanel1.setPositie('l');
    this.configPanel2.setPositie('r');

    this.configPanel1.setTekenPanel(this.tekenPanel1);
    this.configPanel2.setTekenPanel(this.tekenPanel1);
}
```

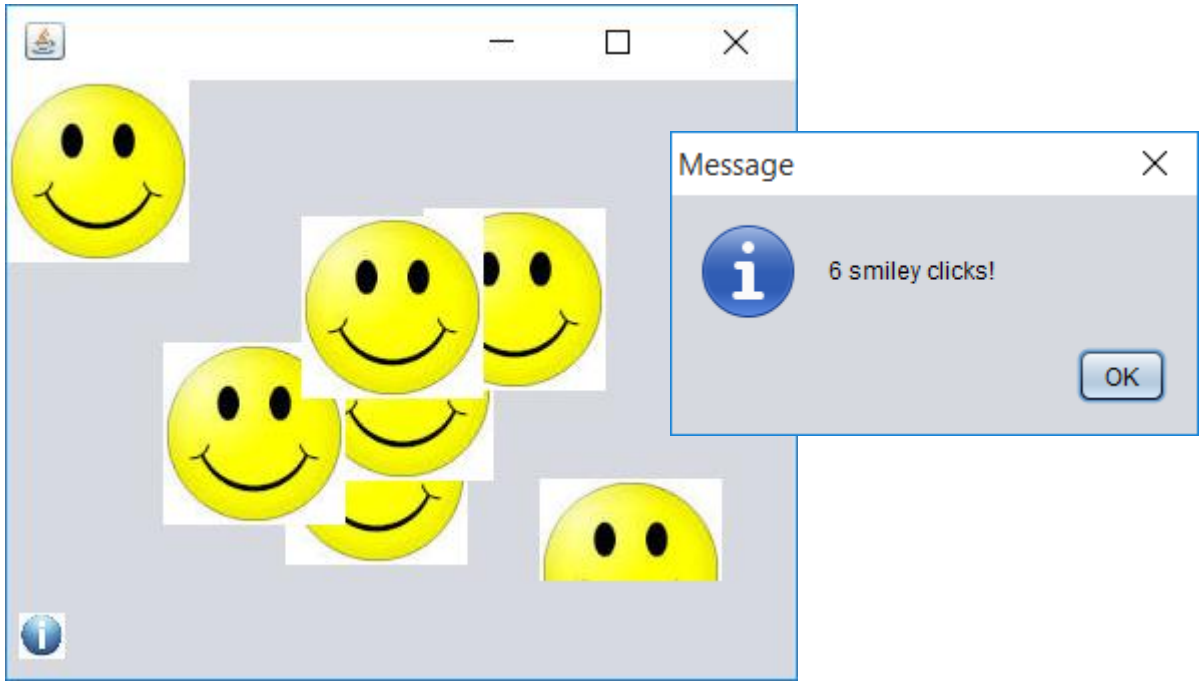
JOptionPane

JOptionPane.showInputDialog
met JOptionPane.QUESTION_MESSAGE

JOptionPane.showMessageDialog
met JOptionPane.ERROR_MESSAGE

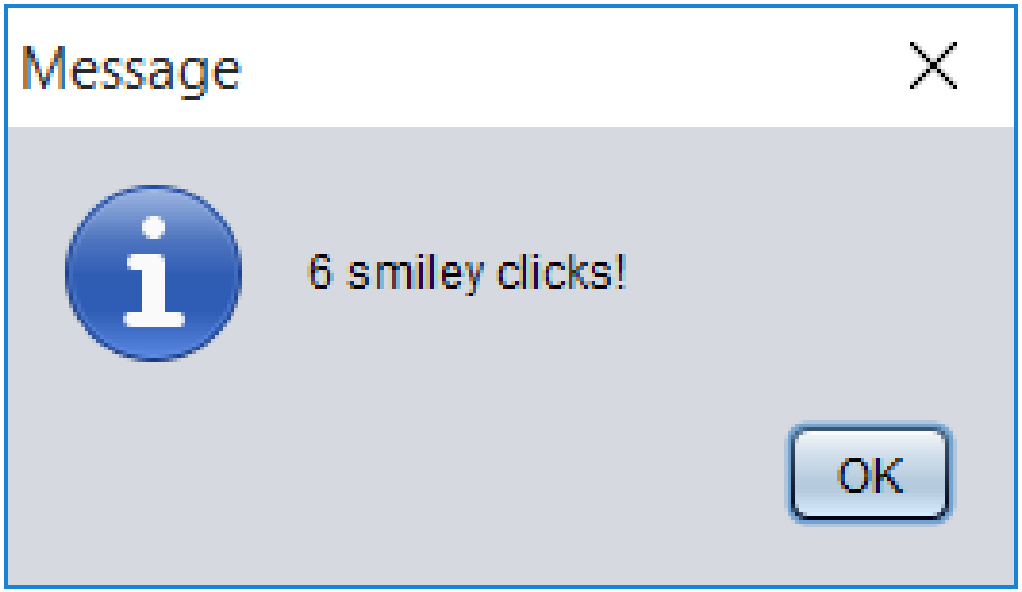
JOptionPane.showOptionDialog
met JOptionPane.QUESTION_MESSAGE

JOptionPane - `messageDialog`




```
private void jLabelInfoMouseClicked(java.awt.event.MouseEvent evt) {
    JOptionPane.showMessageDialog(this, this.newJPanel1.getCount() + " smiley clicks!");
}
```

JOptionPane - `messageDialog`



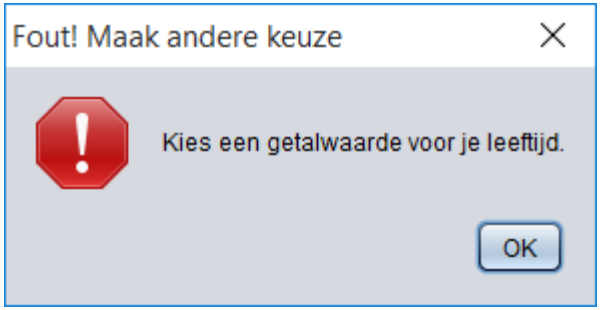

```
public static void showMessageDialog(Component parentComponent,
                                     Object message)
```

JOptionPane - `inputDialog`

```
int dimensie = 0;
do {
   try {
        dimensie = Integer.parseInt(JOptionPane.showInputDialog(this,
                                        "Gelieve de gewenste dimensie op te geven (5-15).",
                                        "Dimensie", JOptionPane.PLAIN_MESSAGE));
   }
   catch(Exception ex) {}
} while (dimensie < 5 || dimensie > 15);
```

Voorbeeld

```java
private void jButtonInputDialogActionPerformed(java.awt.event.ActionEvent evt) {
    int leeftijd = -1;
    int count = 1;
    String vraag = "Wat is je leeftijd?";
    String antwoord ;
    do {
       antwoord = JOptionPane.showInputDialog(this,
                   vraag + (count==0? "" : " (" + count + ")"),
                   "Test meerderjarigheid", JOptionPane.QUESTION_MESSAGE);

       if (antwoord != null) {
               try {
                   leeftijd = Integer.parseInt(antwoord);
               }
               catch (Exception ex) {
                   JOptionPane.showMessageDialog(this,
                           "Kies een getalwaarde voor je leeftijd.",
                           "Fout! Maak andere keuze", JOptionPane.ERROR_MESSAGE
                           /*, new ImageIcon("images/error.jpg")*/);
               }
               count++;
       }
       else {
           break;
       }
    } while ((leeftijd < 0 || leeftijd > 100) && (count <= 5));

    if (leeftijd > 18) {
        JOptionPane.showMessageDialog(this,"MEERDERJARIG!");
    }
}
```



JOptionPane

```
try {
   int leeftijd = Integer.parseInt(JOptionPane.showInputDialog(this,
                                    "Wat is je leeftijd?",
                                    "Test meerderjarigheid",
                                    JOptionPane.QUESTION_MESSAGE) );
 } catch (Exception ex) {
            JOptionPane.showMessageDialog(this,
                "Kies een getalwaarde voor je leeftijd.",
                "FOUT! Maak andere keuze",
                JOptionPane.ERROR_MESSAGE,
                new Icon() );
}
```

Icon is abstract; cannot be instantiated

(Alt-Enter shows hints)

javax.swing.JOptionPane

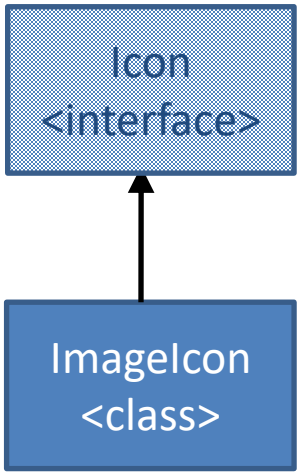
public static void **showMessageDialog**(Component cmpnt, Object o, String string, int i, Icon icon) throws HeadlessException

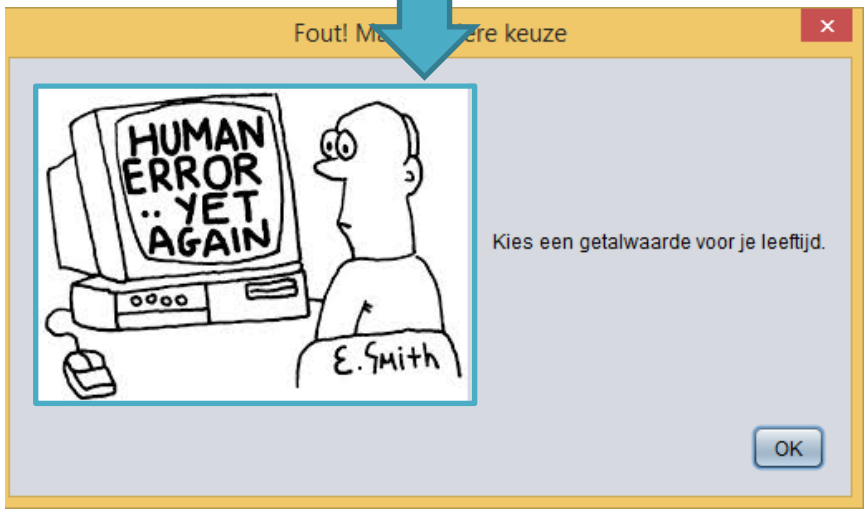
Brings up a dialog displaying a message, specifying all parameters.

Parameters:

- parentComponent - determines the Frame in which the dialog is displayed; if null, or if the parentComponent has no Frame, a default Frame is used
- message - the Object to display
- title - the title string for the dialog
- messageType - the type of message to be displayed: ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, or PLAIN_MESSAGE
- icon - an icon to display in the dialog that helps the user identify the kind of message that is being displayed

```
try {
            ...
 } catch (Exception ex) {
            JOptionPane.showMessageDialog(this,
                "Kies een getalwaarde voor je leeftijd.",
                "FOUT! Maak andere keuze",
                JOptionPane.ERROR_MESSAGE,
                new ImageIcon("images/error.jpg") );
}
```





javax.swing

Class ImageIcon

java.lang.Object
    javax.swing.ImageIcon

All Implemented Interfaces:

Serializable, Accessible, Icon

interface in javax.swing

```
public class ImageIcon
extends Object
implements Icon, Serializable, Accessible
```

Je kan geen object maken van een interface,
wel van een klasse die deze interface implementeert,
(op voorwaarde dat deze klasse op zich niet abstract gedefinieerd is) -- zie eerder--

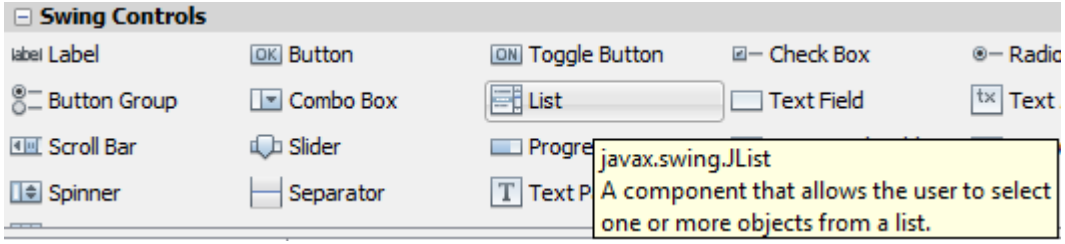
JOptionPane - `optionDialog`

```
public void startOpties() {
    int optie;

    Object[] options = {"Vooringesteld", "Uit database", "Willekeurig"};

    do {
      optie = JOptionPane.showOptionDialog(this, "Gelieve een optie te kiezen:",
                    "Kruiswoordraadsel menu", JOptionPane.YES_NO_CANCEL_OPTION,
                    JOptionPane.QUESTION_MESSAGE, null, options, options[0] );
    } while (optie != 0 && optie != 1 && optie !=2);

    verwerk(optie);
}
```

javax.swing.JList

- grafische component
- Geen OO-verband met java.util.List interface
- Hoe opvullen/raadplegen?
  - Via array
  - Via model (DefaultListModel)

Het `javax.swing.JList` component



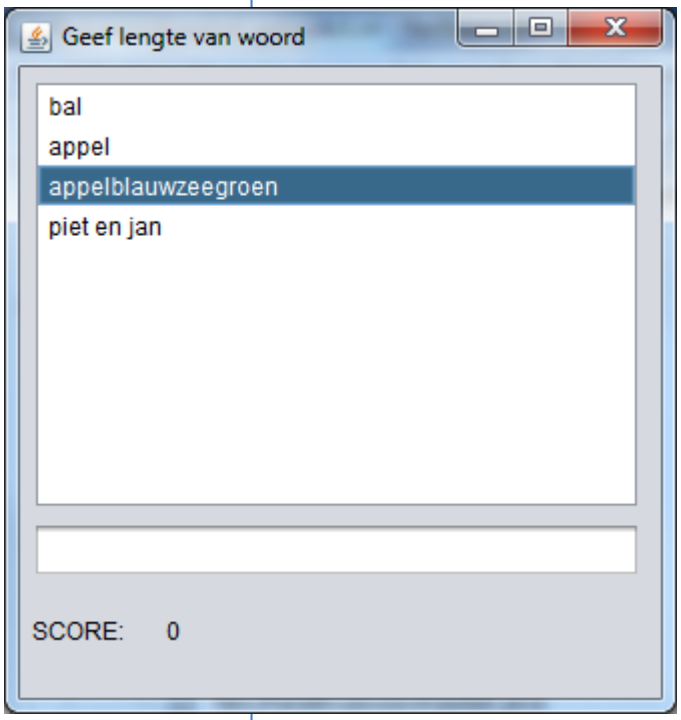
OPTIE1: JList opvullen via array(rij)

```
public class NewJFrame extends javax.swing.JFrame {
   public NewJFrame() {
      super("gevulde JList");
      initComponents();

      String[] rij = {"appel", "peer", "banaan", "pruim"};

      this.jList1.setListData(rij);
   ...
}
```


```
public class NewJFrame extends javax.swing.JFrame {
   public NewJFrame() {
      super("gevulde JList");
      initComponents();

      this.jList1.setListData(new String[] {"appel", "peer", "banaan", "pruim"});
   ...
}
```

JList – opvullen (setListData)

```
public class NewJFrameQuiz extends javax.swing.JFrame {
    public NewJFrameQuiz() {
        super("Geef lengte van woord");
        initComponents();

        String[] vragen = {
            "bal",
            "appel",
            "appelblauwzeegroen",
            "piet en jan"};

        this.jListVragen.setListData(vragen);
        this.jListVragen.setSelectedIndex(2);
    }

    private javax.swing.JList jListVragen;
}
```

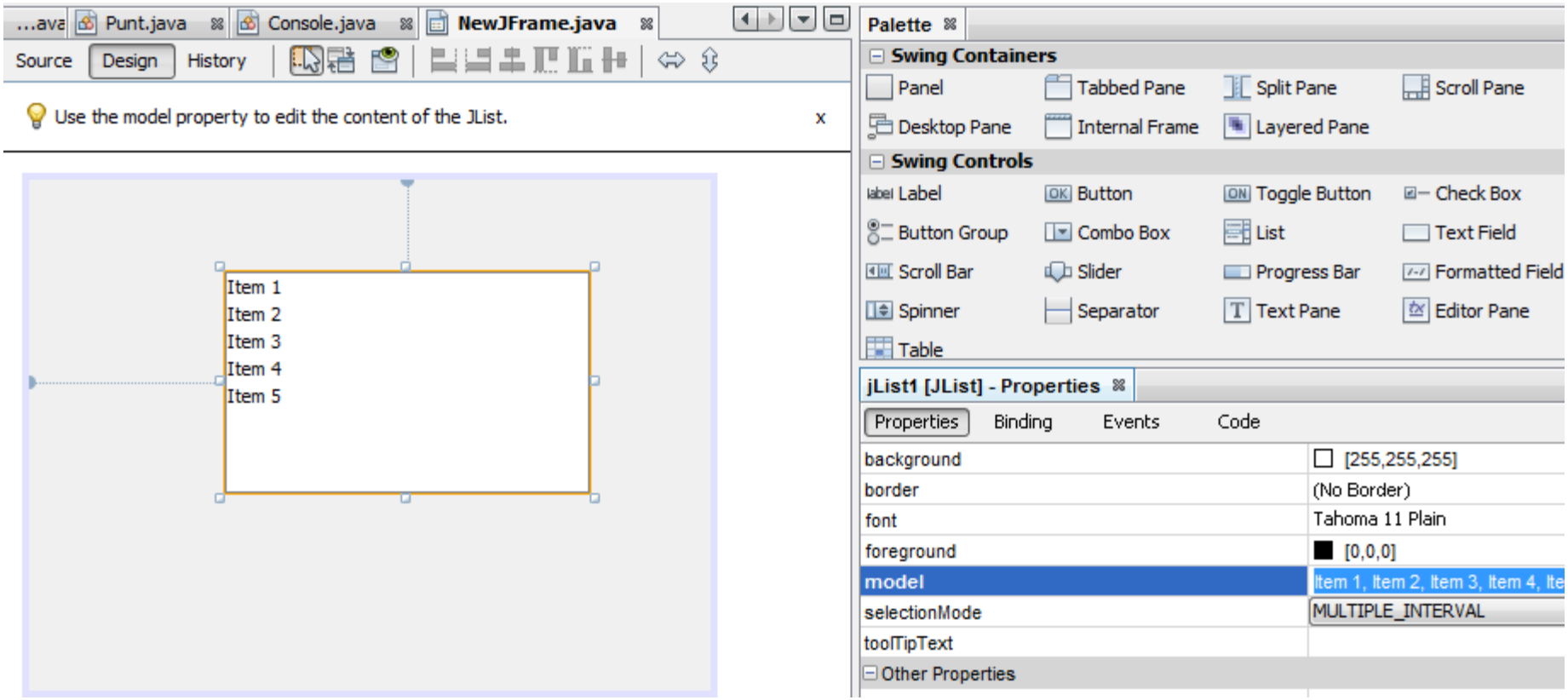


TOEPASSING:
Laat de gebruiker de lengte van het geselecteerde woord ingeven en controleer de juistheid

JList – uitlezen (getSelectedValue)

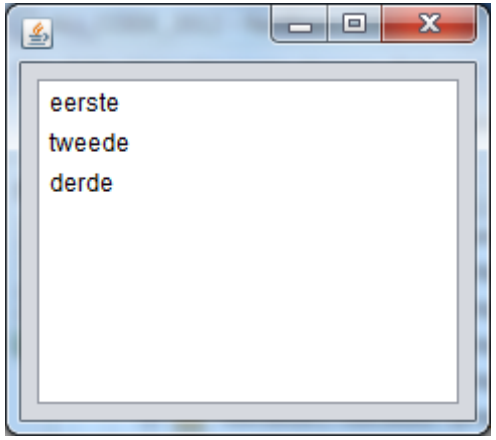
```
public class NewJFrameQuiz extends javax.swing.JFrame {
   private void jTextFieldAntwoordActionPerformed(java.awt.event.ActionEvent evt) {
      String antwoord = this.jTextFieldAntwoord.getText();
      String item = this.jListVragen.getSelectedValue().toString();

      if (item.length() != Integer.parseInt(antwoord)) {
         JOptionPane.showMessageDialog(this, "Foutief antwoord");
      } else {
         this.jLabelScore.setText("" + (Integer.parseInt(this.jLabelScore.getText()) + 1));
      }

      this.jTextFieldAntwoord.setText("");
   }
}
```

OPTIE2: JList opvullen via ‘model’

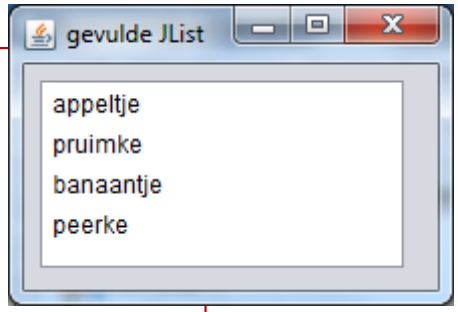


JList – opvullen (setModel i.p.v. setListData)

```
public NewJFrame() {
    initComponents();

    DefaultListModel<String> model = new DefaultListModel<String>();
    model.addElement("eerste");
    model.addElement("tweede");
    model.addElement("derde");

    this.jList1.setModel(model);
}
```



JList opvullen via model




```
public class NewJFrame extends javax.swing.JFrame {
   public NewJFrame() {
      super("gevulde JList");
      initComponents();

      DefaultListModel model = new DefaultListModel();

      model.add(0, "appeltje");
      model.add(1, "peerke");
      model.add(2, "banaantje");
      model.add(3, "pruimke");

      model.addElement("appeltje");
      model.addElement("peerke");
      model.addElement("banaantje");
      model.addElement("pruimke");

      this.jList1.setModel(model);
   }
   …
}
```

Opvullen van een `JList` a.d.h.v. een 'model'

```
DefaultListModel<String> model = new DefaultListModel<String>();
model.add(0, "appeltje");
model.add(1, "peerke");
model.add(2, "banaantje");
model.add(3, "pruimke“);

this.jList1.setModel(model);
```

ListModel is een interface!!!

Het concrete 'model' dat je meegeeft moet een object zijn van een klasse die deze interface implementeert!

Interface & implementerende klasse

javax.swing

Interface ListModel

All Known Subinterfaces:

ComboBoxModel, MutableComboBoxModel

All Known Implementing Classes:

AbstractListModel, BasicDirectoryModel, DefaultComboBoxModel, DefaultListModel, MetalFileChooserUI.DirectoryComboBoxModel, MetalFileChooserUI.FilterComboBoxModel

~~ListModel model = new ListModel();~~

javax.swing

Class AbstractListModel

```
java.lang.Object
  └ javax.swing.AbstractListModel
```

All Implemented Interfaces:

Serializable, ListModel

Direct Known Subclasses:

BasicDirectoryModel, DefaultComboBoxModel, DefaultListModel, MetalFileChooserUI.DirectoryComboBoxModel, MetalFileChooserUI.FilterComboBoxModel

```
public abstract class AbstractListModel
extends Object
implements ListModel, Serializable
```

~~AbstractListModel model = new AbstractListModel();~~

Interface & implementerende klasse

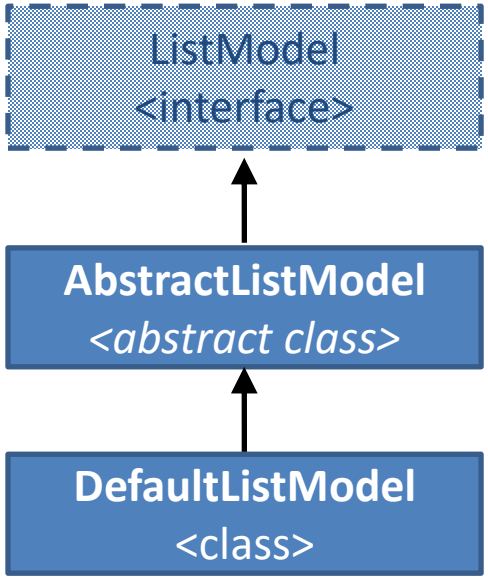
javax.swing

Class DefaultListModel

```
java.lang.Object
  └ javax.swing.AbstractListModel
      └ javax.swing.DefaultListModel
```

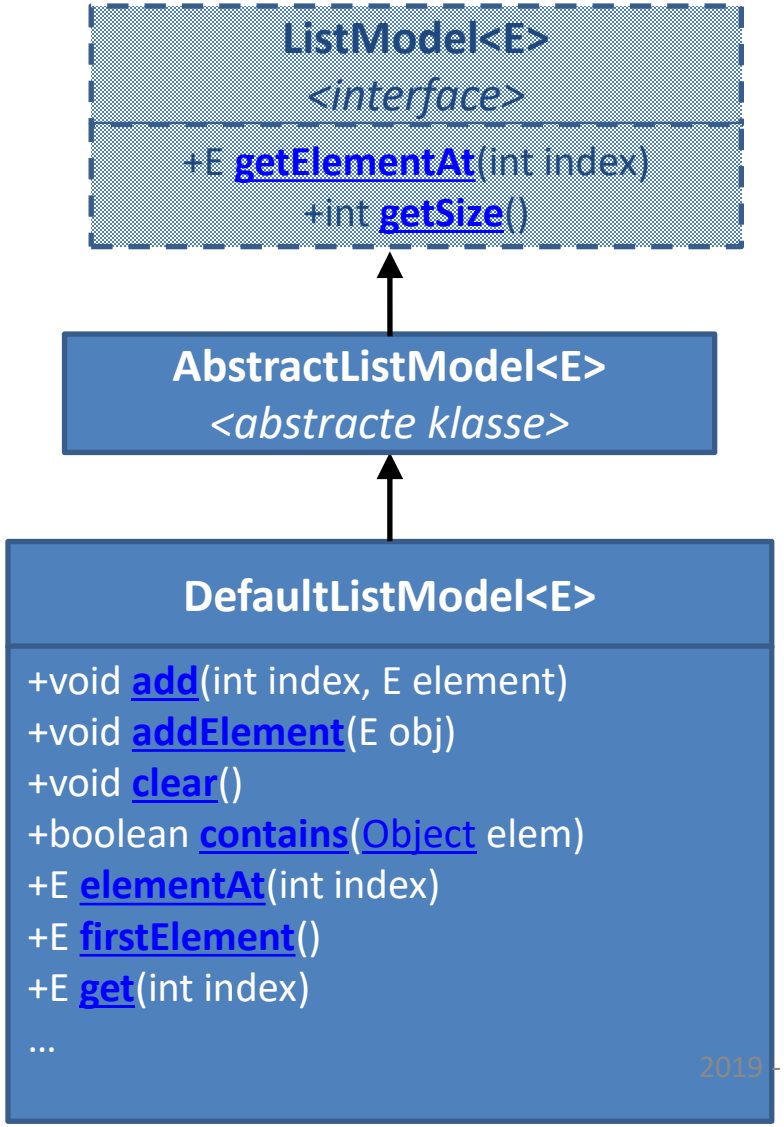
All Implemented Interfaces:
Serializable, ListModel

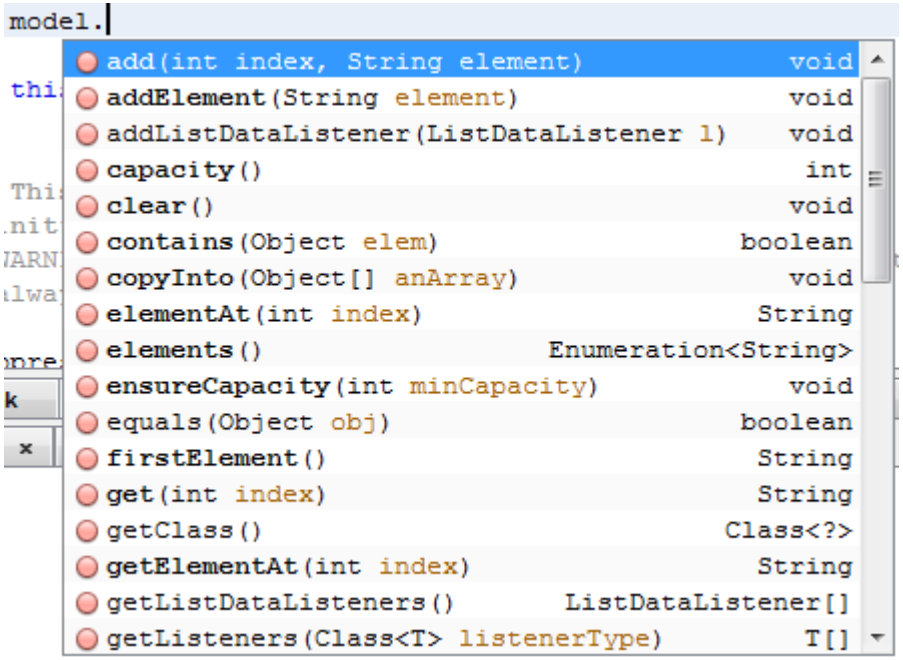
```
public class DefaultListModel
extends AbstractListModel
```



DefaultListModel model = new DefaultListModel();

DefaultListModel<E>

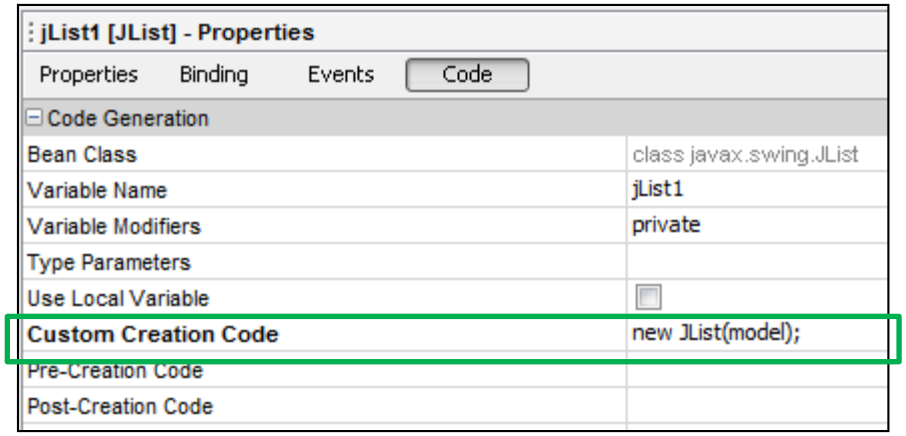




‘model’ toegepast op labo12

```
try {
        quiz = new Quiz(naam);

        DefaultListModel model = new DefaultListModel();
        for (String s : quiz.geefAlleVragen()) {
           model.addElement(s);
        }

        this.jListVragen.setModel(model);

} catch (IOException ex) {
        JOptionPane.showMessageDialog(this, ex);
}
```

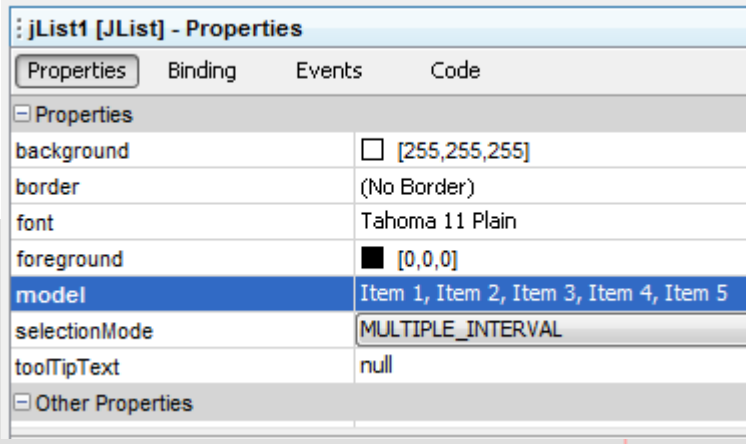
Alternatief: model via JList properties venster: Custom Creation Code



in GUI:

```
public class NewJFrame extends javax.swing.JFrame {
   private DefaultListModel model = new DefaultListModel(); //veld in de klasse

   public NewJFrame() {
      initComponents();

      model.clear();
      model.addElement("appeltje");
      model.addElement("peerke");
      model.addElement("banaantje");
      model.addElement("pruimke");
}
```

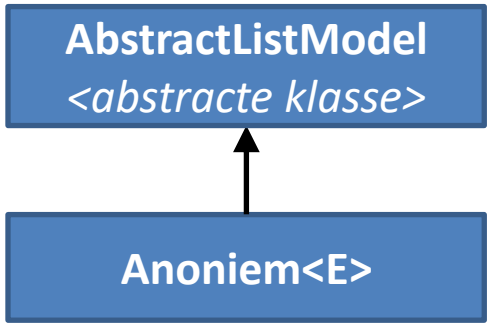
Gegenereerde code i.g.v. opvulling via JList Properties venster




```
private void initComponents() {

    jScrollPane1 = new javax.swing.JScrollPane()
    jList1 = new javax.swing.JList();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jList1.setModel(new javax.swing.AbstractListModel() {
        String[] strings = { "Item 1", "Item 2", "Item 3", "Item 4", "Item 5" };
        public int getSize() { return strings.length; }
        public Object getElementAt(int i) { return strings[i]; }
    });
```

Object van anonieme inner klasse als parameter!

<u>Noot:</u> Die anonieme inner klasse is een subklasse van de abstracte klasse AbstractListModel

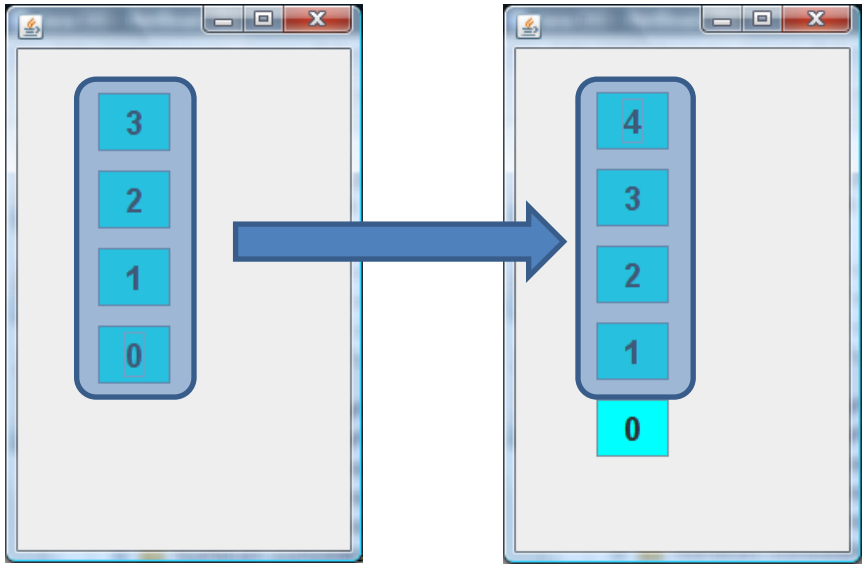


Even piepen in de *javax.swing.JList* klasse

```
public void setListData(final E[] listData) {
    setModel (
        new AbstractListModel<E>() {
            public int getSize() { return listData.length; }
            public E getElementAt(int i) { return listData[i]; }
        }
    );
}
```

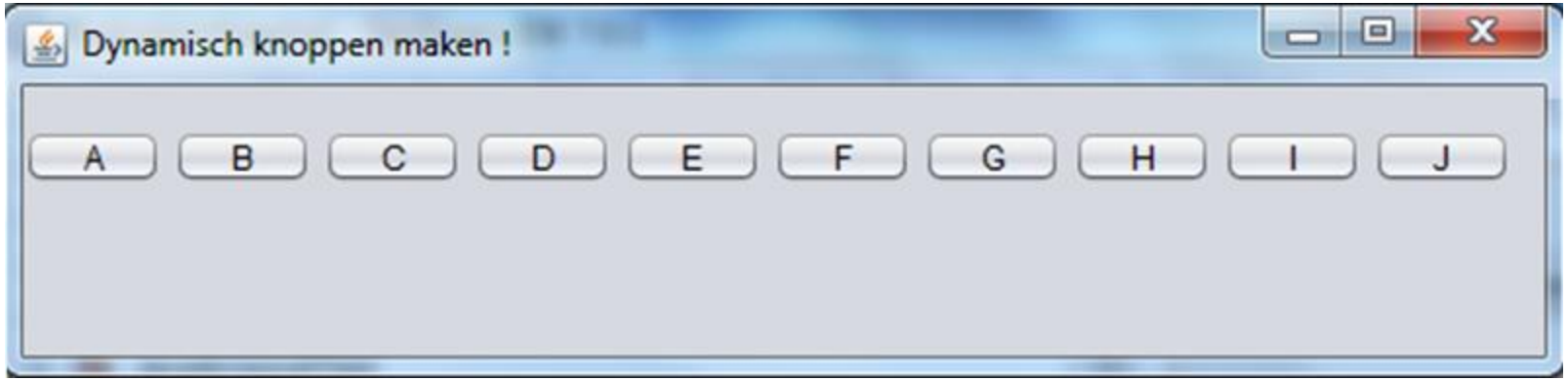
Dynamisch werken met grafische componenten

Designer venster:



```java
public class VensterBis extends javax.swing.JFrame {
  public VensterBis() {
        initComponents();
        definieerKnopTekst();
  }

  private void definieerKnopTekst() {
        Component[] c = this.getContentPane().getComponents();

        for(int i = 0; i < c.length; i++) {
            if (c[i] instanceof JButton) {
                c[i].setFont(new Font("Courier", Font.BOLD, 20));
                c[i].setBackground(Color.CYAN);

                ((JButton)c[i]).setText(Integer.toString(i));
            }
        }
  }

  ...
}
```

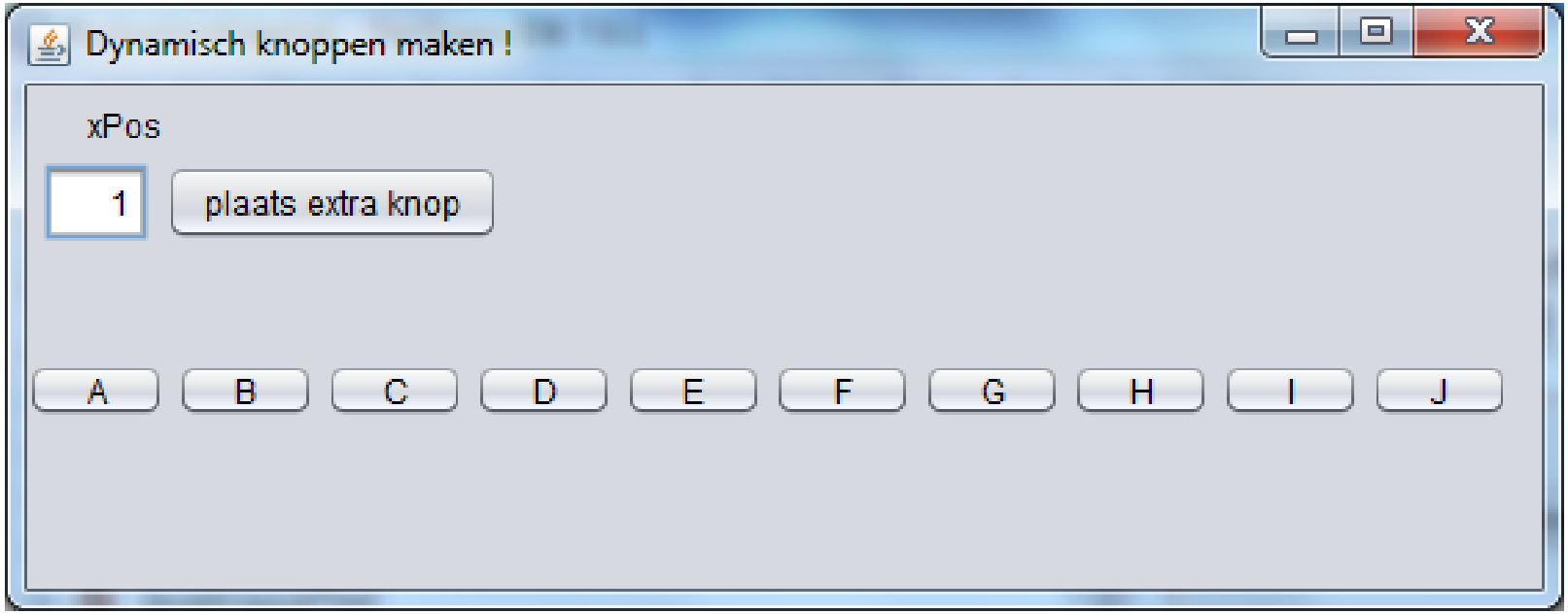
Grafische componenten aanmaken via code



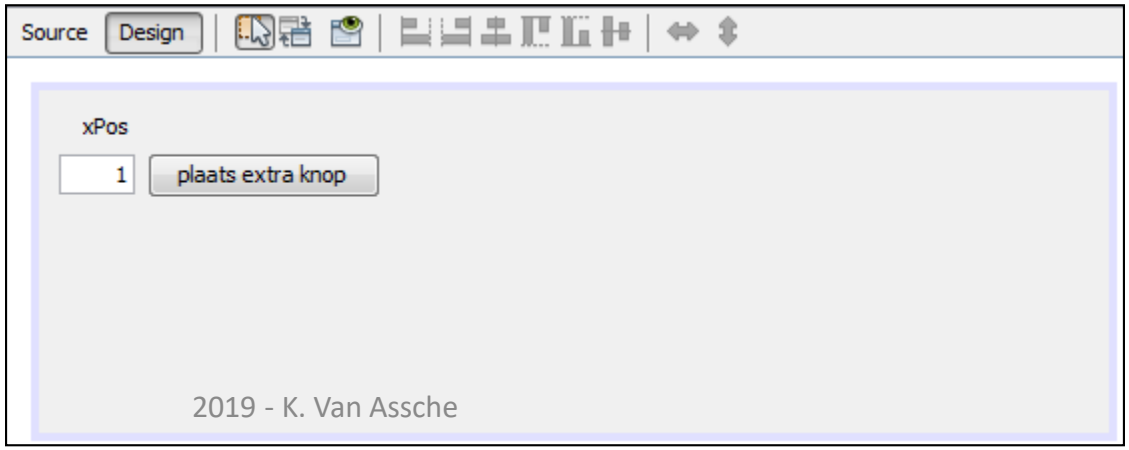
Hard gecodeerd

```
public class Knoppenrij extends javax.swing.JFrame {

    /** Creates new form Knoppenrij */
    public Knoppenrij() {
        super("Dynamisch knoppen maken !");

        initComponents();
        initExtraComponents();
    }

    /**...*/
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {...}// </editor-fold>

    private void initExtraComponents() {
        for (int i = 0 ; i < 10 ; i++) {
            JButton knopje = new JButton("" + (char)('A' + i));
            knopje.setBounds(0, 0, 50, 20);
            knopje.setLocation(54 * i, 100);
            this.add(knopje);
        }
    }
```

Interactief grafische componenten bijplaatsen



Corresponderende Designer view:



Knoppen bijplaatsen via GUI-interactie


```
private void jButtonExtraknopActionPerformed(java.awt.event.ActionEvent evt) {
        JButton knopje = new JButton(this.jTextFieldXpos.getText());
        knopje.setBounds(0, 0, 50, 20);
        knopje.setLocation(54 * Integer.parseInt(this.jTextFieldXpos.getText()), 120);
        this.add(knopje);
        this.repaint();
}
```

EVENT HANDLING VIA CODE

Event handling via code (i.p.v. via designer venster)

```
public class Knoppenrij extends javax.swing.JFrame {
   private JButton[] knoppenrij;

    public Knoppenrij() {
      initComponents();
      initKnoppen(10);
      toonKnoppenInFrame();
      actieKnoppen();
   }

   …
```

```
private void initKnoppen(int aantal) {
    knoppenrij = new JButton[aantal];
    for (int i = 0 ; i < aantal ; i++) {
        JButton knopje = new JButton("knop" + i);
        knopje.setBounds(0, 0, 100, 20);
        knopje.setLocation(100, 25 * i);
        knoppenrij[i] = knopje;
    }
}
```

```
private void toonKnoppenInFrame() {
    for (JButton knop : knoppenrij) {
        this.add(knop);
    }
}
```

Terzijde: This anonymous inner class creation can be turned into a lambda expression

```
private void actieKnoppen() {
    for (JButton knop : knoppenrij) {
        knop.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent ae) {
                JOptionPane.showMessageDialog(null, ((JButton)ae.getSource()).getText());
            }
        });
    }
}
```

Terzijde: This anonymous inner class creation can be turned into a lambda expression

```
private void actieKnoppen() {
    for (JButton knop : knoppenrij) {
        knop.addActionListener( (ActionEvent ae) -> {
            JOptionPane.showMessageDialog(null, ((JButton)ae.getSource()).getText());
        });
    }
}
```

```
private void actieKnoppen() {
    for (JButton knop : knoppenrij) {
        knop.addActionListener( (ae) -> {
            JOptionPane.showMessageDialog(null, ((JButton)ae.getSource()).getText());
        });
    }
}
```

ActionListener interface

Method Summary

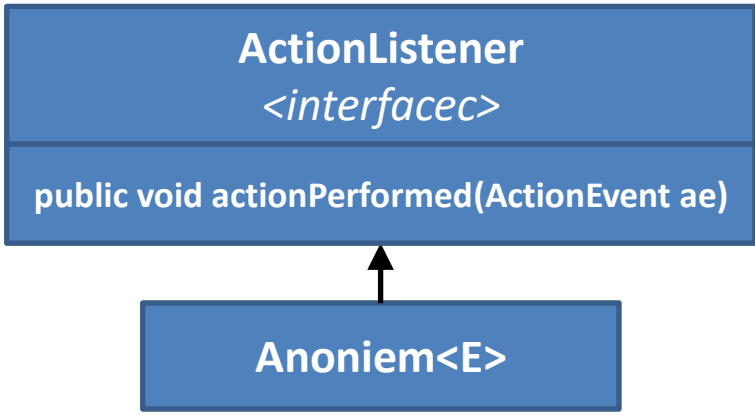
java.awt.event

Interface ActionListener

Methods

Modifier and Type	Method and Description
void	`actionPerformed(ActionEvent e)` Invoked when an action occurs.

```
knop.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            //doeiets
        }
});
```



ActionEvent

java.awt.event

Class ActionEvent

java.lang.Object
    java.util.EventObject
        java.awt.AWTEvent
            java.awt.event.ActionEvent

All Implemented Interfaces:

Serializable

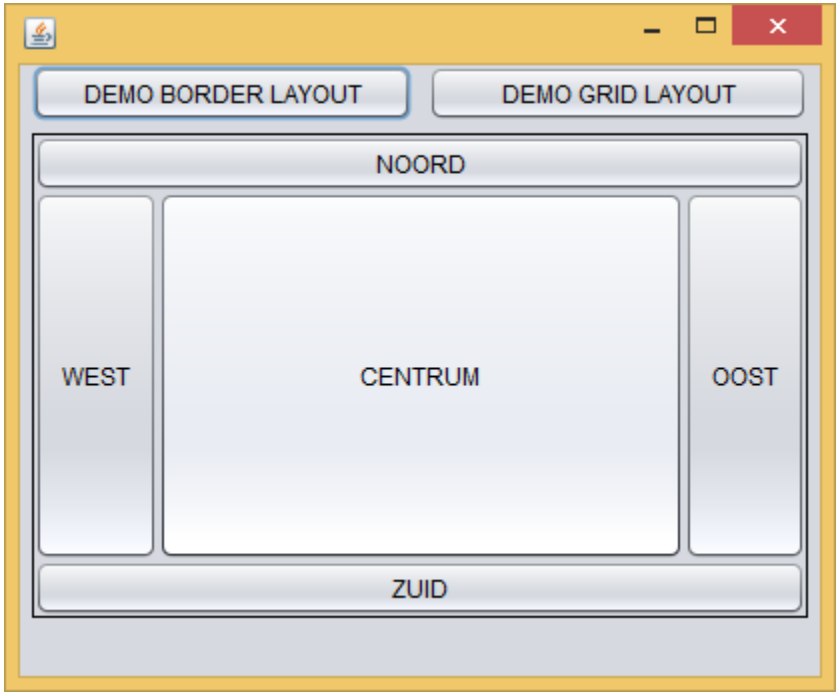
```
public class ActionEvent
extends AWTEvent
```

A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a `Button`) when the component-specific action occurs (such as being pressed). The event is passed to every `ActionListener` object that registered to receive such events using the component's `addActionListener` method.

Even vertaald:

Een Event-object wordt gegenereerd telkens wanneer je op de knop drukt.
Dit event wordt via de actionPerformed methode doorgegeven
aan elk ActionListener-object dat geregistreerd is om deze events te ontvangen
(via de addActionListener methode van de knop)

Layout voor panels
BorderLayout - GridLayout - …



this.jPanel1.**setLayout**(new BorderLayout());

this.jPanel1.**setLayout**(new GridLayout(0,2));

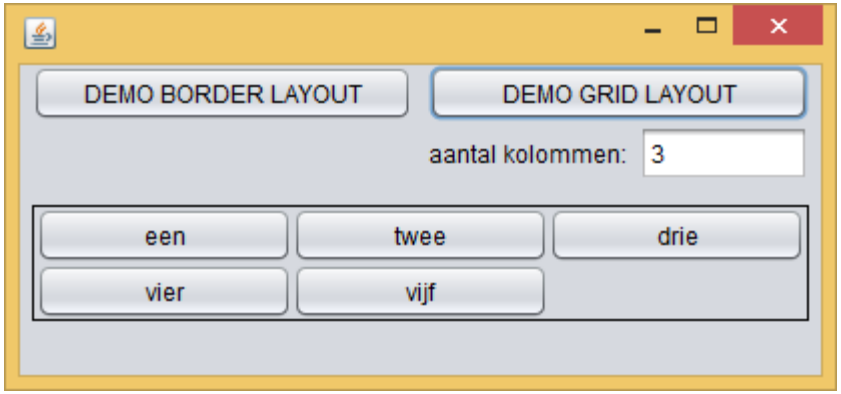
<u>Demo:</u>
Dynamisch instelbaar aantal kolommen

BorderLayout

```
private void jButtonBorderLayoutActionPerformed(java.awt.event.ActionEvent evt) {
    this.jPanel1.removeAll();
    this.jPanel1.setLayout(new BorderLayout());
    this.jPanel1.add(new JButton("een"), BorderLayout.NORTH);
    this.jPanel1.add(new JButton("twee"), BorderLayout.EAST);
    this.jPanel1.add(new JButton("drie"), BorderLayout.SOUTH);
    this.jPanel1.add(new JButton("vier"), BorderLayout.WEST);
    this.jPanel1.add(new JButton("vijf"), BorderLayout.CENTER);
    this.jPanel1.revalidate();
  }
```

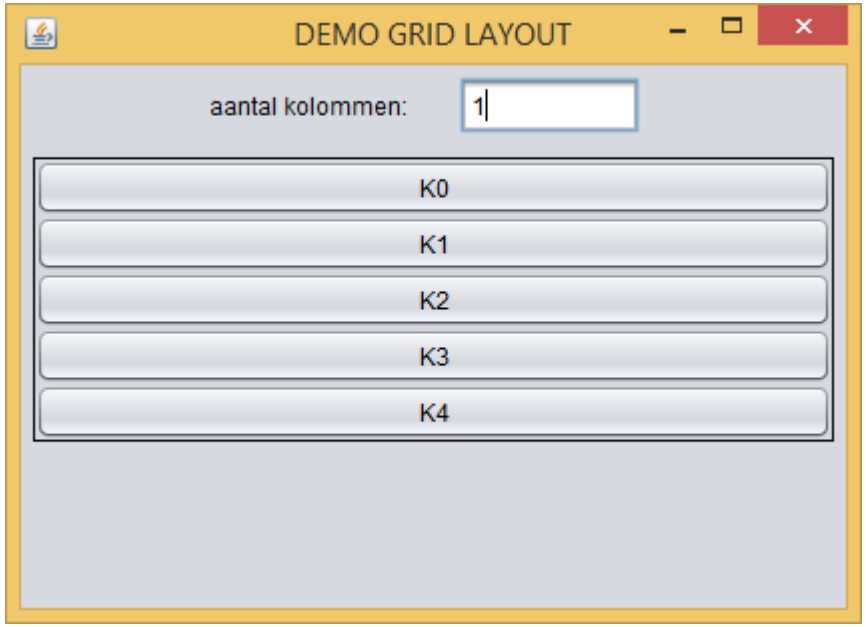
GridLayout

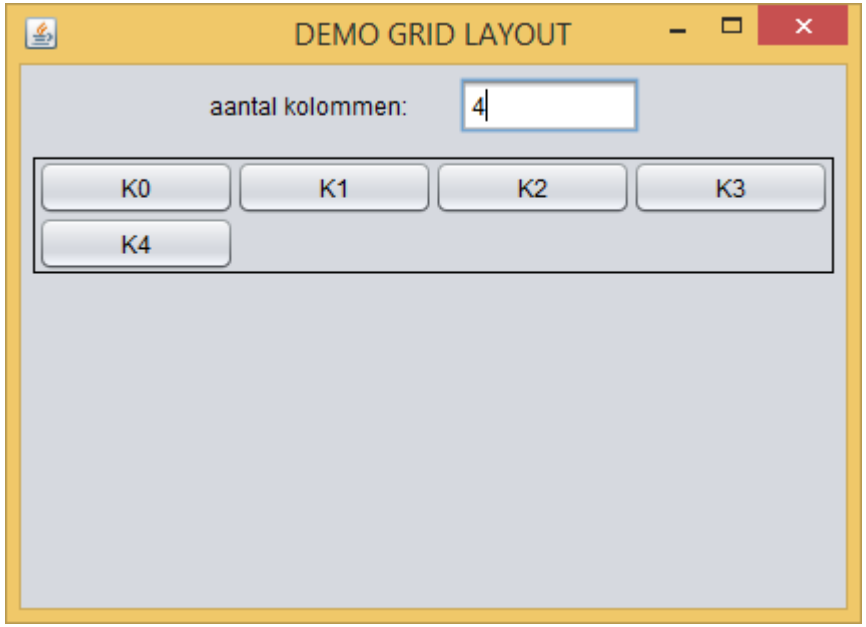
```
private void jButtonGridLayoutActionPerformed(java.awt.event.ActionEvent evt) {
    int aantalKolommen = Integer.parseInt(this.jTextFieldKol.getText());
    this.jPanel1.setLayout(new GridLayout(0,aantalKolommen));
    this.jPanel1.revalidate();
  }
```

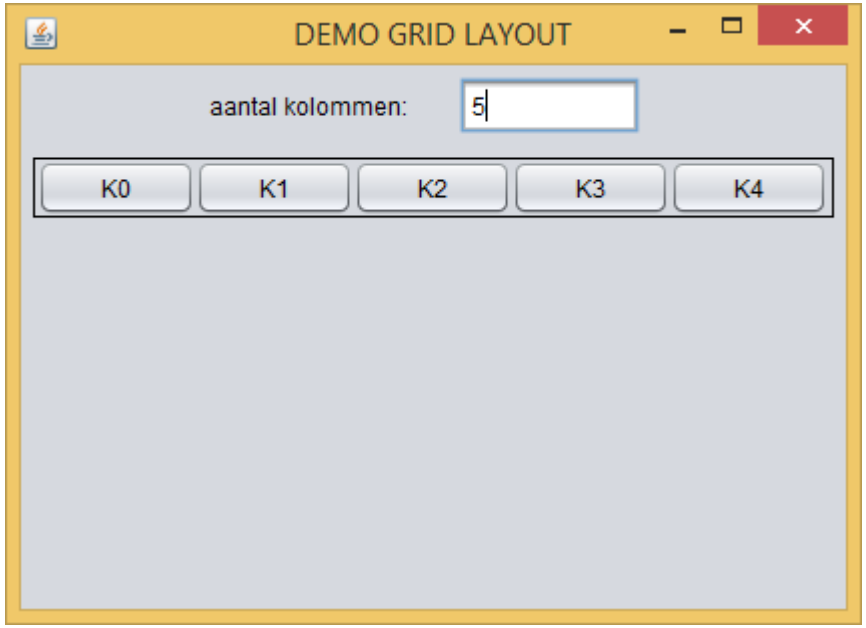


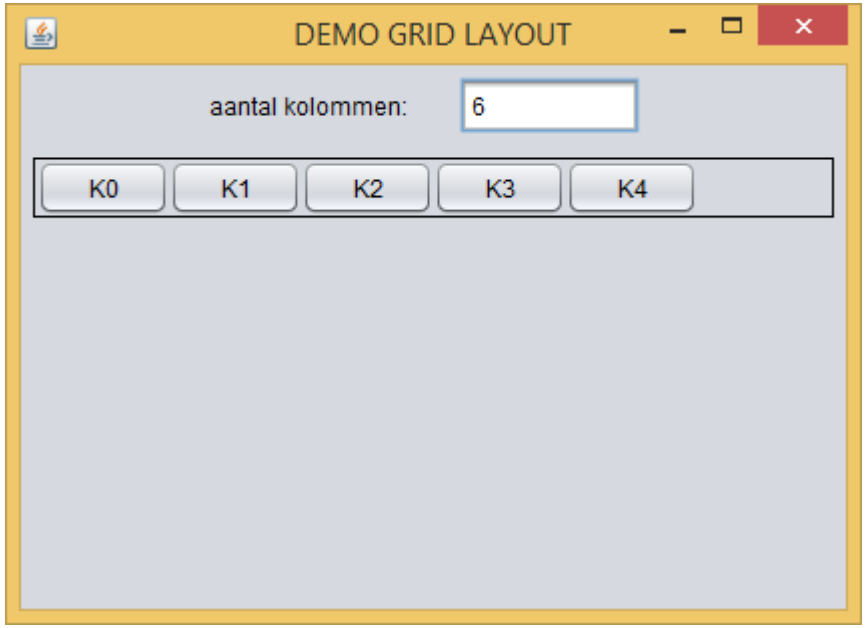
```java
public class DemoGridLayout extends javax.swing.JFrame {
    private JButton[] knoppenrij;
    private static final int AANTAL = 5;

    public DemoGridLayout() {
        super("DEMO GRID LAYOUT");
        initComponents();

        knoppenrij = new JButton[AANTAL];
        for(int i = 0; i < knoppenrij.length; i++) {
            knoppenrij[i] = new JButton("K" + i);
            this.jPanel1.add(knoppenrij[i]);
        }
    }
```

```java
private void jTextFieldKolActionPerformed(java.awt.event.ActionEvent evt) {
    int aantalKolommen = Integer.parseInt(this.jTextFieldKol.getText());
    this.jPanel1.setLayout(new GridLayout(0,aantalKolommen));
    this.jPanel1.revalidate();
}
```


DEMO GRID LAYOUT
aantal kolommen:
1
K0
K1
K2
K3
K4

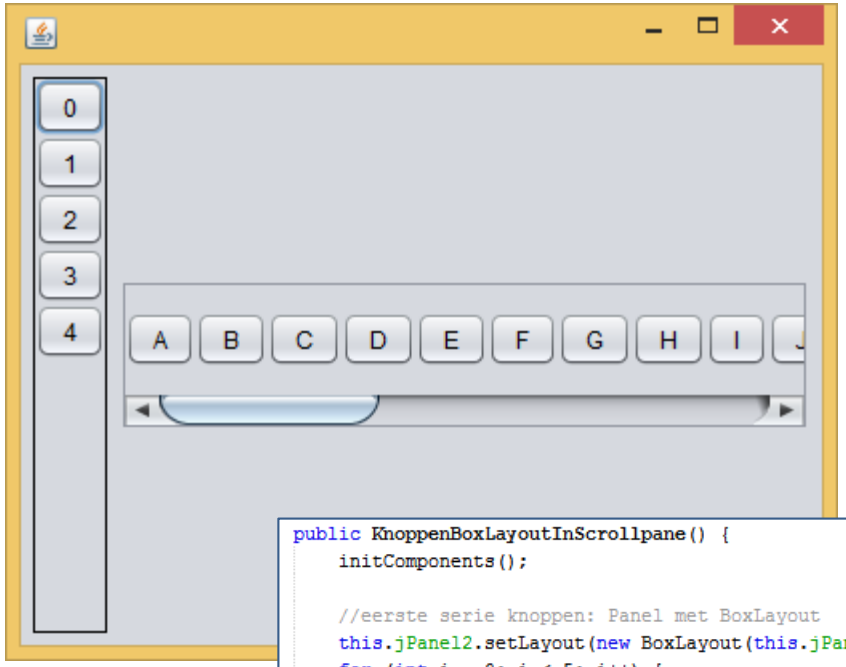

DEMO GRID LAYOUT
aantal kolommen:
4
K0
K1
K2
K3
K4

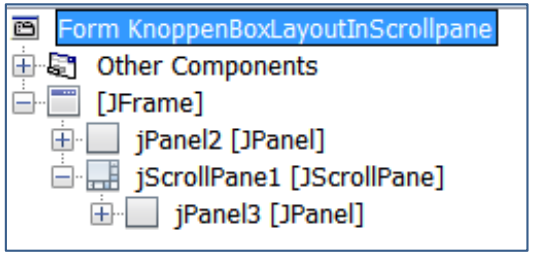

DEMO GRID LAYOUT
aantal kolommen:
5
K0
K1
K2
K3
K4


DEMO GRID LAYOUT
aantal kolommen:
6
K0
K1
K2
K3
K4

BoxLayout

horizontaal of vertikaal, al dan niet in scrollpane

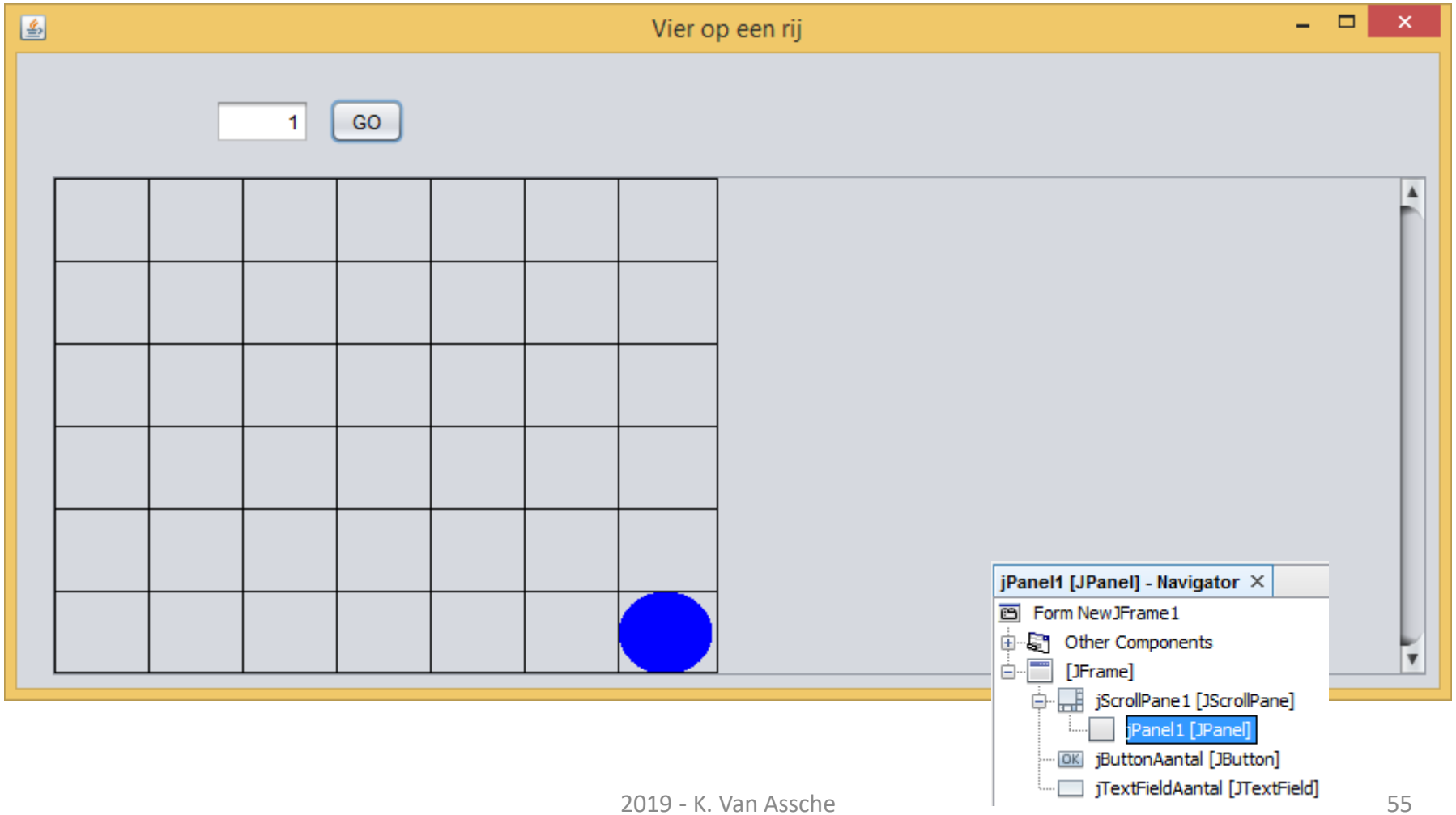


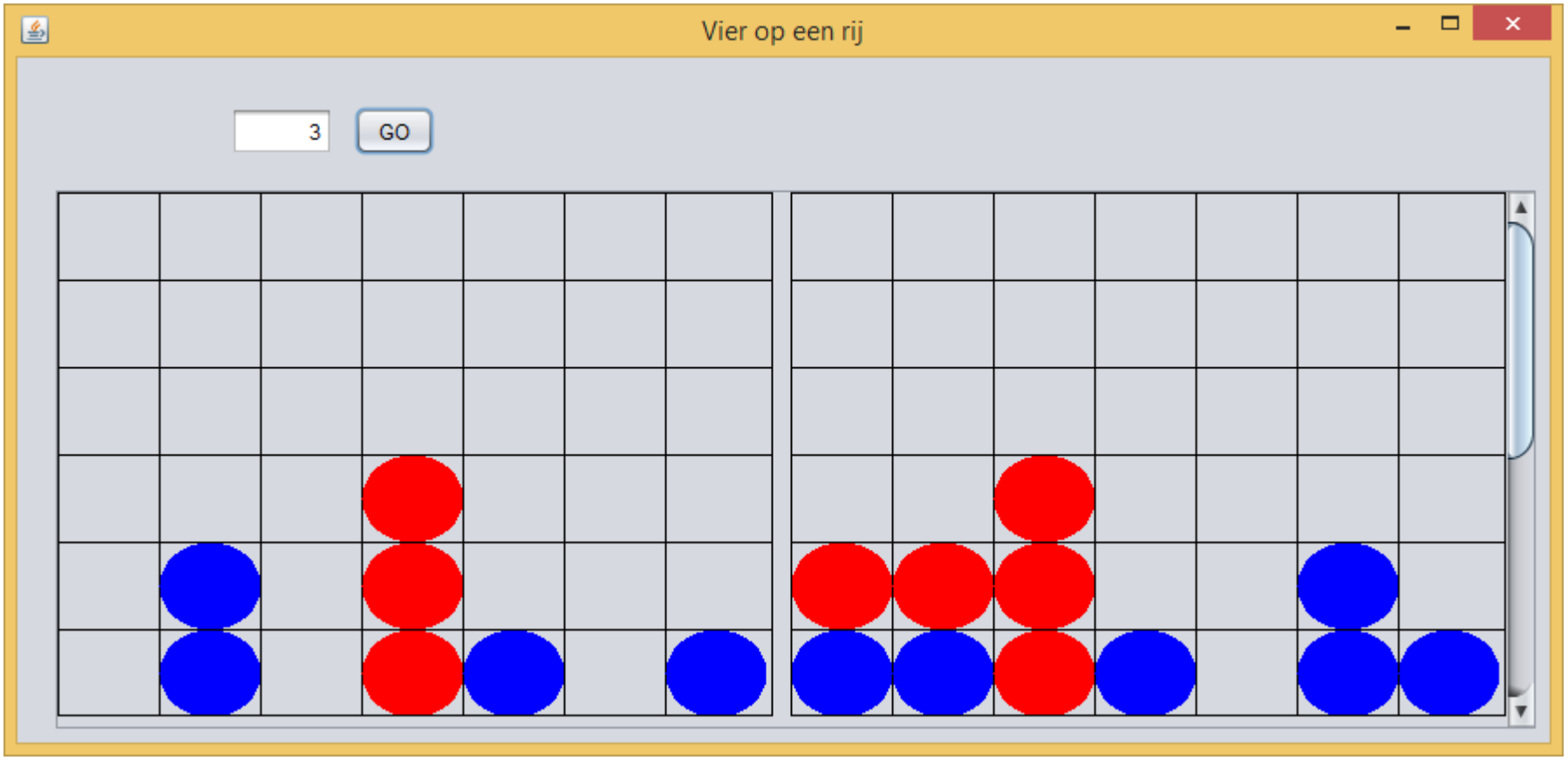



```java
public KnoppenBoxLayoutInScrollpane() {
    initComponents();

    //eerste serie knoppen: Panel met BoxLayout
    this.jPanel2.setLayout(new BoxLayout(this.jPanel2, BoxLayout.Y_AXIS));
    for (int i = 0; i < 5; i++) {
        this.jPanel2.add(new JButton("" + i));
    }

    //tweede serie knoppen: Panel met BoxLayout. Panel zelf is in ScrollPane geplaatst
    this.jPanel3.setLayout(new BoxLayout(this.jPanel3, BoxLayout.X_AXIS));
    for (int i = 0; i < 26; i++) {
        this.jPanel3.add(new JButton("" + (char)('A' + i)));
    }
}
```

Andere layout binnen scrollpane?

* altijd ScrollPaneLayout voor scrollpane component
* andere layouts binnen scrollpane enkel mogelijk door centraal panel in scrollpane te plaatsen en hiervoor de gewenste layout te bepalen






```
this.jPanel1.setLayout(new GridLayout(0, 2, 10, 10));
```